Attorney Docket No. 42P15770 *PATENT*

UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR PERFORMING EFFICIENT TRANSFORMATIONS WITH HORIZONTAL ADDITION AND SUBTRACTION

INVENTORS:

WILLIAM W. MACY
ERIC DEBES
MINERVA YEUNG
YEN-KUANG CHEN
PATRICE ROUSSEL

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(408) 720-8300

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number **EV 339 922 984 US**
Date of Deposit **June 30, 2003**
I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231
Nikia M. Mc-Nillion
(Typed or printed name of person mailing paper or fee)

(Signature of person mailing paper or fee) Date

METHOD AND APPARATUS FOR PERFORMING EFFICIENT TRANSFORMATIONS WITH HORIZONTAL ADDITION AND SUBTRACTION

RELATED APPLICATIONS

[0001] This is a continuation-in-part application claiming, under 35 U.S.C. § 120, the benefit of the filing dates of U.S. application Ser. No. 09/952,891, filed Oct. 29, 2001, currently pending; and of U.S. application Ser. No. 10/193,645, filed July 9, 2002, currently pending; which is a continuation of application Ser. No. 9/053,401, filed March 31, 1998, now US Patent 6,418,529.

FIELD OF THE DISCLOSURE

[0002] This disclosure relates generally to the field of processors. In particular, the disclosure relates to calculation of Single-Instruction-Multiple-Data (SIMD) horizontal addition and subtraction operations.

BACKGROUND OF THE DISCLOSURE

[0003] To improve the efficiency of multimedia applications, as well as other applications with similar characteristics, a Single Instruction, Multiple Data (SIMD) architecture has been implemented in computer systems to enable one instruction to operate on several operands simultaneously, rather than on a single operand. In particular, SIMD architectures take advantage of packing many data elements within one register or memory location. With parallel hardware execution, multiple operations can be performed on

separate data elements with one instruction, resulting in significant performance improvement.

[0004] One set of SIMD instructions was defined for the Pentium® Processor with MMX™ Technology by Intel® Corporation and described in "IA-32 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference," which is available from Intel Corporation, Santa Clara, CA on the world-wide-web (www) at intel.com/design/litcentr.

[0005] Currently, the SIMD addition operation only performs "vertical" or inter-register addition, where pairs of data elements, for example, a first element Xn (where n is an integer) from one operand, and a second element Yn from a second operand, are added together. An example of such a vertical addition operation is shown in Figure 1, where the instruction is performed on the sets of data elements (X_3, X_2, X_1 and X_0) and (Y_3, Y_2, Y_1, and Y_0) accessed as Source1 and Source2, respectively to obtain the result (X_3+Y_3, X_2+Y_2, X_1+Y_1, and X_0+Y_0).

[0006] Although many applications currently in use can take advantage of such a vertical add operation, there are a number of important applications which would require the rearrangement of the data elements before the vertical add operation can be implemented so as to provide realization of the application.

[0007] For example, an 8-point decimation in time operation of a Walsh-Hadamard transform and of a Fast-Fourier Transform (FFT) is shown in Figure 2b. The larger 8-point transforms may be performed in stages through successive doubling. That is to say an 8-point transform can be computed from two 4-point transforms, which can be computed from four 2-point transforms. The computations at each stage are called butterflies.

[0008] A butterfly for the staged computations of Figure 2b is shown in Figure 2a. At each successive stage, data elements at even positions are combined with the data elements at odd positions to generate data elements of the next successive stage. In order to perform these staged computations using prior-art SIMD vertical additions and vertical subtractions, instructions substantially similar to the instruction sequence example of Table 1 may be used to shuffle and rearrange data elements for each stage.

Exemplary Code To Prepare Data for Vertical-Add/Vertical-Subtract Operations:

```
movdqa      xmm7, [esi]      //shuffle pattern to put even elements in low half odd in high half
pshufb      xmm0, xmm7       //shuffle data in xmm0
pshufb      xmm1, xmm7       //shuffle data in xmm1
movdqa      xmm2, xmm0       //copy xmm0 data
punpcklqdq  xmm0, xmm1       //combine even elements of xmm0 and xmm1. xmm1 in high half.
punpckhqdq  xmm2, xmm1       //combine odd elements of xmm2 (equal to xmm0) and xmm1.
```

Table 1

[0009] One drawback of this approach is that it requires additional processing time to perform the operations that shuffle and recombine the data elements between stages. Another drawback is that an additional register is used to hold a shuffle pattern for sorting the even elements into the low half of the register and the odd elements into the high half of the register. A third drawback is that the extra instructions that are required due to the necessity to rearrange data between stages reduces the code density and requires more storage in memory and in cache.

[0010] Accordingly, there is a need in the technology for providing an apparatus and method which more efficiently performs butterfly computations, such as those used in 2-point or 4-point transforms for example, without requiring additional time to perform operations that shuffle and recombine data elements. There is also a need in the technology for a method and operation for increasing code density by eliminating the necessity for the

rearrangement of data elements and thereby eliminating the corresponding rearrangement operations from the code. By eliminating the necessity for the rearrangement of data elements, an additional register could also be made available that might otherwise have been used to store patterns for shuffling the odd and even data elements.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings.

[0012] **Figure 1** illustrates the vertical or inter-add operation of the prior art.

[0013] **Figure 2a** illustrates an example of a butterfly for staged computations of a Walsh-Hadamard transform and of a Fast-Fourier Transform (FFT).

[0014] **Figure 2b** illustrates an example of an 8-point decimation in time operation of a Walsh-Hadamard transform and of a Fast-Fourier Transform (FFT).

[0015] **Figure 3a** illustrates one embodiment of a horizontal or intra-add operation.

[0016] **Figure 3b** illustrates one embodiment of a horizontal or intra-subtract operation.

[0017] **Figure 3c** illustrates an alternative embodiment of a horizontal or intra-add operation.

[0018] **Figure 3d** illustrates another alternative embodiment of a horizontal or intra-add-subtract operation.

[0019] **Figures 4a-4l** illustrate one embodiment of a transformation using horizontal or intra-addition/subtraction operations for performing the butterfly stages.

[0020] **Figure 5a** illustrates an example of a computer system in accordance with one embodiment of the invention.

[0021] **Figure 5b** illustrates another example of a computer system in accordance with an alternative embodiment of the invention.

[0022] **Figure 5c** illustrates another example of a computer system in accordance with an alternative embodiment of the invention.

[0023] **Figure 6a** is a depiction of an operation encoding (opcode) format for a horizontal or intra-addition/subtraction instruction.

[0024] **Figure 6b** is a depiction of an alternative opcode format for a horizontal or intra-addition/subtraction instruction.

[0025] **Figure 6c** is a depiction of another alternative opcode format for a horizontal or intra-addition/subtraction instruction.

[0026] **Figure 7a** illustrates packed data-types in accordance with one embodiment of the invention.

[0027] **Figure 7b** illustrates packed data-types in accordance with an alternative embodiment of the invention.

[0028] **Figure 8a** is a flow diagram illustrating one embodiment of a process for performing the intra-add operation of Figure 3a.

[0029] **Figure 8b** is a flow diagram illustrating one embodiment of a process for performing the intra-subtract operation of Figure 3b.

[0030] **Figure 8c** is a flow diagram illustrating an alternative embodiment of a process for performing the intra-add operation.

[0031] **Figure 8d** is a flow diagram illustrating an alternative embodiment of a process for performing the intra-subtract operation.

[0032] **Figure 8e** is a flow diagram illustrating one alternative embodiment of a process for performing the intra-add operation of Figure 3c.

[0033] **Figure 8f** is a flow diagram illustrating another alternative embodiment of a process for performing the intra-subtract operation.

[0034] **Figure 8g** is a flow diagram illustrating one alternative embodiment of a process for performing the intra-add operation of Figure 3a.

[0035] **Figure 8h** is a flow diagram illustrating one alternative embodiment of a process for performing the intra-subtract operation of Figure 3b.

[0036] **Figure 9a** illustrates one embodiment of a circuit for performing horizontal or intra-add/subtract operations.

[0037] **Figure 9b** illustrates an alternative embodiment of a circuit for performing horizontal or intra-add/subtract operations.

DETAILED DESCRIPTION

[0038] Disclosed herein is a process and apparatus for performing horizontal intra-add operations on packed data. One embodiment of a processor performs operations on data elements in a first packed data to generate a plurality of data elements in a second packed data in response to receiving an instruction. At least two of the plurality of data elements in the second packed data store the results of a horizontal intra-add operation, at least one of these results coming from the operation on data elements of the first packed data. One embodiment of a software process utilizes horizontal intra-add instructions for efficiently performing butterfly computations as may be employed, for example, in Walsh-Hadamard transforms or in Fast-Fourier Transforms.

[0039] These and other embodiments of the present invention may be realized in accordance with the following teachings and it should be evident that various modifications and changes may be made in the following teachings without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense and the invention measured only in terms of the claims and their equivalents. Some well-known circuits, structures and techniques may not be shown in detail in order not to obscure the invention.

[0040] For the purpose of the following discussion of embodiments of the present invention, illustrative terms are used. Definitions for certain such illustrative terms follows.

[0041] A computer system or data processing device or system may be understood to mean any one of a variety of devices or systems for accessing data and/or communications. Examples include but are not limited to any combinations of one or more of the following:

laptop computers, notebook computers; desktop computers, personal digital assistants, handheld computers, personal organizers; palmtop computers, pocket computers, cellular telephone/fax devices, game computers, digitizing tablet devices, electronic books, and digital video or digital audio recorder/players.

[0042] A register is any device capable of storing and providing data. Further functionality of a register with respect to data formats is described below. A register is not necessarily, included on the same die or in the same package as the processor.

[0043] A wireless device or interface may be understood to mean any one of a variety of devices or interfaces for wireless communications. Examples include but are not limited to any combination of devices for one or more of the following: short-range radio, satellite communications, wireless local area networks, wireless telephony, cellular digital packet data, home radio frequency, narrowband time-division multiple access, code-division multiple access, wideband code-division multiple access, wireless fidelity or short message service.

[0044] One aspect of the present invention is a process for, or a processor including instructions for performing horizontal or intra-addition operations on packed data. In one embodiment, two pairs of data elements (e.g., X_3 and X_2, and X_1 and X_0) located within a single storage area (e.g., Source1) are added together using a horizontal or an intra-add operation. In alternate embodiments, data elements from each of two storage areas (e.g., Source1 and Source2) are added/subtracted and stored as data elements of a resulting packed data, for example, as shown in **Figures 3a-3d**.

[0045] Another aspect of the present invention involves a method and apparatus for performing butterfly stages, for example, in Walsh-Hadamard transforms or in Fast-Fourier

Transforms, using a horizontal or intra-addition/subtraction operation. For example, **Figure 2a** illustrates a butterfly for the staged computations of such a transformation. At each stage of an N-point transformation, the outputs of the butterfly, F(k) and F(k+N/2) are computed from the inputs $F_{even}(k)$ and $F_{odd}(k)$ as an averaged weighted sum or difference as follows:

$$F(k) = (1/2)\bullet(F_{even}(k) + F_{odd}(k)\bullet W_N^K),$$

$$F(k+N/2) = (1/2)\bullet(F_{even}(k) - F_{odd}(k)\bullet W_N^K).$$

The weights, W_N^K, may very by stage depending on the type of transform. For example, in the 8-point Walsh-Hadamard transform, W_8^K is equal to one. It will be appreciated that multiplications by one half (1/2) may also be accomplished through shift operations, for example, at each stage or shifting the weights prior to multiplying or shifting results after several stages.

[0046] **Figure 2b** illustrates an 8-point decimation in time operation of a Walsh-Hadamard transform and of a Fast-Fourier Transform (FFT). Values shown along the horizontal lines represent multiplication. For example f_4 is multiplied by the value of weight, W_8^0, in Stage 1 as illustrated with respect to the second horizontal line of Figure 2b. The arrows from one horizontal line to another horizontal line at each stage signify additions or subtractions, which are further illustrated through multiplication by negative one. It will be appreciated that in practice, such operations may be accomplished through a variety of different methods, for example, multiplication by negative weight values followed by addition, multiplication by positive weight values followed by subtraction, etc. Further details of performing the transformation illustrated using intra-addition/subtraction are discussed with respect to Figures 4a-4l. It will be appreciated that transformations of

conveniently chosen sizes may be carried out through the use of butterfly stages in a manner substantially similar to the 8-point transformation illustrated in Figure 2b.

[0047] Through the use of horizontal or intra-addition/subtraction the butterfly stages of, for example, a Walsh-Hadamard transform may be efficiently computed. In combination with SIMD multiplication the butterfly stages of, for example, a Fast-Fourier Transform may also be efficiently computed. **Figure 3a** illustrates one embodiment of the horizontal or intra-add operation. A first operand, Source1, comprises four data elements, (X_3, X_2, X_1 and X_0). A second operand, Source2, also comprises four data elements (Y_3, Y_2, Y_1, and Y_0). In response to an intra-add instruction, a result is produced comprising four data elements respectively representing the operations, (Y_2+Y_3, Y_0+Y_1, X_2+X_3, and X_0+X_1). It will be appreciated that horizontal or intra-add operations may be performed on more (e.g. 6, 8, 10, 16, etc.) or less data elements that illustrated in Figure 3a. It will also be appreciated that the order of data elements in a result may be varied, for example representing the operations, (Y_0+Y_1, Y_2+Y_3, X_0+X_1, and X_2+X_3) without departing from the broader spirit and scope of the invention.

[0048] It will also be appreciated that the sizes of the data elements may be conveniently chosen (e.g. 1-byte, 2-bytes, 4-bytes, 17-bits, 23-bits, 64-bits, etc.) according to the particularities of the data and/or algorithms employed. Further, such operands may comprise floating-point data, fixed-point signed data, unsigned data, binary-coded-decimal data, carry-save redundant encoded data, sign-digit redundant encoded data, etc.

[0049] **Figure 3b** illustrates one embodiment of the horizontal or intra-subtract operation. As in the previous example, a first operand, Source1, comprises four data

elements, (X_3, X_2, X_1 and X_0). A second operand, Source2, also comprises four data elements (Y_3, Y_2, Y_1, and Y_0). In response to an intra-subtract instruction, a result is produced comprising four data elements respectively representing the operations, (Y_2-Y_3, Y_0-Y_1, X_2-X_3, and X_0-X_1). For an alternative embodiment of the intra-subtract instruction, a result may be produced comprising data elements representing the operations, (Y_3-Y_2, Y_1-Y_0, X_3-X_2, and X_1-X_0).

[0050] **Figure 3c** illustrates an alternative embodiment of the horizontal or intra-add operation. A first operand, Source1, comprises two data elements, (X_1 and X_0). A second operand, Source2, also comprises two data elements (Y_1, and Y_0). In response to an intra-subtract instruction, a result is produced comprising two data elements respectively representing the operations, (Y_0+Y_1, and X_0+X_1). Similarly, in response to an intra-subtract instruction, a result may be produced comprising two data elements respectively representing the operations, (Y_0-Y_1, and X_0-X_1).

[0051] **Figure 3d** illustrates another alternative embodiment of the horizontal or intra-add-subtract operation. As in the example of Figures 3a and 3b, a first operand, Source1, comprises four data elements, (X_3, X_2, X_1 and X_0), and a second operand, Source2, comprises four data elements (Y_3, Y_2, Y_1, and Y_0). In response to an intra-add-subtract instruction, a result is produced comprising four data elements respectively representing the operations, (Y_2+Y_3, Y_1-Y_0, X_2+X_3, and X_1-X_0). For an alternative embodiment of the intra-add-subtract instruction, a result may be produced comprising data elements representing the operations, (Y_2-Y_3, Y_0+Y_1, X_2-X_3, and X_0+X_1).

[0052] It will be appreciated that while the following example illustrates an 8-point transformation using intra-add/subtract operations similar to those illustrated in Figures 3a and 3b. Other transformation examples may be more readily adapted to an alternative embodiment of intra-addition/subtraction.

[0053] In one embodiment of a transformation, each 16-bit data element from a first source is multiplied with corresponding 16-bit data elements from a second source, for example, as shown in Figures 4a & 4b, generating two 64-bit intermediate results, each of which are stored in separate storage areas. Intra-add and intra-subtract operations are performed on each of the intermediate results to generate a plurality of data elements, which are stored as packed results, for example, as shown in Figures 4c & 4d.

[0054] In an alternative embodiment of a transformation, each 32-bit data element from a first source is multiplied with corresponding 32-bit data elements from a second source, as shown in Figures 4a & 4b, generating two 128-bit intermediate results, each of which are stored in separate storage areas. Intra-add and intra-subtract operations are then performed on each of the 128-bit intermediate results to generate a plurality of 32-bit data elements, which are stored as packed results, as shown in Figures 4c & 4d. For one embodiment of a transformation, each data element is a fixed-point number or an integer. For an alternative embodiment of a transformation, each data element is a floating-point number.

[0055] Now turning to the example illustrated, and following the computations with respect to the 8-point transformation illustrated in Figure 2b, it will be appreciated that Figures 4a-4d compute the four butterflies of Stage 1.

[0056] For example in **Figure 4a**, a first operand, Inputs1, comprises four data elements, (f_6, f_2, f_4 and f_0), and a second operand, Weight1, comprises four coefficients ($W_8^0/2$, 1/2,

$W8^0/2$, and 1/2). In response to a SIMD multiply instruction, an intermediate result, IResult1, is produced comprising four data elements respectively representing the operations, ($f6 \cdot W8^0/2$, $f2/2$, $f4 \cdot W8^0/2$, and $f0/2$). Likewise in **Figure 4b**, a first operand, Inputs2, comprises four data elements, ($f7$, $f3$, $f5$ and $f1$), and the second operand, Weight1, comprises the four coefficients ($W8^0/2$, 1/2, $W8^0/2$, and 1/2). In response to a second SIMD multiply instruction, a second intermediate result, IResult2, is produced comprising four data elements respectively representing the operations, ($f7 \cdot W8^0/2$, $f3/2$, $f5 \cdot W8^0/2$, and $f1/2$).

[0057] Then in **Figure 4c**, IResult1 and IResult2 are combined in response to an intra-add instruction, to produce a result comprising four data elements respectively representing the operations, $((f3+f7 \cdot W8^0)/2$, $(f1+f5 \cdot W8^0)/2$, $(f2+f6 \cdot W8^0)/2$, and $(f0+f4 \cdot W8^0)/2)$, which correspond to the input elements of Stage2 of Figure 2b, (r_3, r_2, r_1 and r_0) respectively. In **Figure 4d**, IResult1 and IResult2 are combined in response to an intra-subtract instruction, to produce a result comprising four data elements respectively representing the operations, $((f3-f7 \cdot W8^0)/2$, $(f1-f5 \cdot W8^0)/2$, $(f2-f6 \cdot W8^0)/2$, and $(f0-f4 \cdot W8^0)/2)$, which correspond to the input elements of Stage2 of Figure 2b, (s_3, s_2, s_1 and s_0) respectively. It will be appreciated that for one embodiment of the operations of each stage only three registers may be required thereby permitting multiple transformations to be executed in parallel or larger transformations on more data elements.

[0058] In **Figure 4e**, the Stage 2 input elements, (r_3, r_2, r_1 and r_0), and Weight1, ($W8^0/2$, 1/2, $W8^0/2$, and 1/2) are combined in response to a SIMD multiply instruction, to generate a third intermediate result, IResult3, comprising four data elements respectively representing the operations, ($r3 \cdot W8^0/2$, $r2/2$, $r1 \cdot W8^0/2$, and $f0/2$). In **Figure 4f**, Stage 2 input elements,

(s_3, s_2, s_1 and s_0), and a second operand, Weight2, comprising four coefficients ($W8^2/2$, 1/2, $W8^2/2$, and 1/2) are combined in response to a SIMD multiply instruction to generate a fourth intermediate result, IResult4, comprising four data elements respectively representing the operations, ($s3 \cdot W8^2/2$, $s2/2$, $s1 \cdot W8^2/2$, and $s0/2$).

[0059] Then in **Figure 4g**, IResult3 and IResult4 are combined in response to an intra-add instruction, to produce a result comprising four data elements respectively representing the operations, (($s2+s3 \cdot W8^2$)/2, ($s0+s1 \cdot W8^2$)/2, ($r2+r3 \cdot W8^0$)/2, and ($r0+r1 \cdot W8^0$)/2), which correspond to the input elements of Stage 3 of Figure 2b, (t_3, t_2, t_1 and t_0) respectively. In **Figure 4h**, IResult3 and IResult4 are combined in response to an intra-subtract instruction, to produce a result comprising four data elements respectively representing the operations, (($s2-s3 \cdot W8^2$)/2, ($s0-s1 \cdot W8^2$)/2, ($r2-r3 \cdot W8^0$)/2, and ($r0-r1 \cdot W8^0$)/2), which correspond to the input elements of Stage 3 of Figure 2b, (u_3, u_2, u_1 and u_0) respectively.

[0060] In **Figure 4i**, these Stage 3 input elements, (t_3, t_2, t_1 and t_0), and a second operand, Weight3, comprising four coefficients ($W8^1/2$, 1/2, $W8^0/2$, and 1/2) are combined in response to a SIMD multiply instruction, to generate a fifth intermediate result, IResult5, comprising four data elements respectively representing the operations, ($t3 \cdot W8^1/2$, $t2/2$, $t1 \cdot W8^0/2$, and $t0/2$). In **Figure 4j**, Stage 3 input elements, (u_3, u_2, u_1 and u_0), and a second operand, Weight4, comprising four coefficients ($W8^3/2$, 1/2, $W8^2/2$, and 1/2) are combined in response to a SIMD multiply instruction to generate a sixth intermediate result, IResult6, comprising four data elements respectively representing the operations, ($u3 \cdot W8^3/2$, $u2/2$, $u1 \cdot W8^2/2$, and $u0/2$).

[0061] Finally in **Figure 4k**, IResult5 and IResult6 are combined in response to an intra-add instruction, to produce a result comprising four data elements respectively representing the operations, $((u_2+u_3 \cdot W8^3)/2$, $(u_0+u_1 \cdot W8^2)/2$, $(t_2+t_3 \cdot W8^1)/2$, and $(t_0+t_1 \cdot W8^0)/2)$, which correspond to the desired output elements of Figure 2b, (F_3, F_2, F_1 and F_0) respectively. In **Figure 4l**, IResult5 and IResult6 are combined in response to an intra-subtract instruction, to produce a result comprising four data elements respectively representing the operations, $((u_2-u_3 \cdot W8^3)/2$, $(u_0-u_1 \cdot W8^2)/2$, $(t_2-t_3 \cdot W8^1)/2$, and $(t_0-t_1 \cdot W8^0)/2)$, which correspond to the output elements of Figure 2b, (F_7, F_6, F_5 and F_4) respectively.

[0062] For alternative embodiments of a transformation, additional SIMD instructions may also be useful (e.g. SIMD shift operations for adjusting fixed-point data). For one alternative embodiment of a transformation, vertical or inter-add/subtract operations may also be used in conjunction with horizontal or intra-add/subtract operations, for example, in subsequent stages to perform transformations on more than eight input values. For another embodiment of a transformation, subsequent applications of separable transformations may be used to perform a higher dimensional transformation. It will also be appreciated that while the transformation illustrated uses a radix-2 algorithm, other algorithms (e.g. radix-4 or split-radix) may be used instead.

[0063] One embodiment of Figures 4a-4l illustrates an example using an integer data format in which the result register may be of the same size as the source register(s). In alternative embodiments, the width of the data stored in a result register may differ from that of the source register(s), for example, to store the full width of the products generated in Figures 4a & 4b in a result register. In other alternative embodiments a floating point data format may be used. In alternative embodiments, the width of the packed data stored in the

source register(s) and the result register may comprise 128-bits, or 256-bits, 320-bits or some other conveniently chosen size. In addition, although the discussions above pertain to packed operands that have four data elements, alternative embodiments may involve packed operands that have at least two data elements (i.e., that are double wide).

COMPUTER SYSTEM

[0064] **Figure 5a** illustrates one embodiment of a computer system 100 which implements the principles of the present invention. Computer system 100 comprises a bus 102 for communicating information, and a processor 110 for processing information. In one embodiment, the bus 102 may be any communications hardware and/or software for communicating information. The processor 110 represents a central processing unit of any type of architecture, examples of which include a CISC, a RISC or a VLIW type architecture. Processor 110 may be suitable for manufacture in one or more process technologies and by being represented on a machine readable media in sufficient detail, may be suitable to facilitate said manufacture.

[0065] Computer system 100 further comprises a main memory 104 that is coupled to bus 102, for storing information and instructions to be executed by the processor 110. Computer system 110 also comprises a read only memory (ROM) 106 and/or other status storage device, coupled to the bus 102 for storing information and instructions for access and execution by processor 110. In addition, computer system 110 comprises a data storage device 108 that is coupled to the bus 102 for storing information and instructions.

[0066] As shown in Figure 5a, processor 110 comprises an execution unit 120, a set of register file(s) 130, a cache memory 140, a decoder 150 and an internal bus 160. The

processor 110 also includes additional circuitry (not shown) which is not necessary to the understanding of the present invention.

[0067] Execution unit 120 is used for executing instructions received by processor 110. In addition to recognizing instructions typically implemented in general purpose processors, execution unit 120 recognizes instructions in packed instruction set 122 for performing operations on packed data formats. Packed instruction set 122 includes instructions for supporting intra-add/subtract and multiply operations. In addition, packed instruction set 122 may also include other packed instructions.

[0068] Execution unit 120 is coupled to register file 130 by internal bus 160. Register file 130 represents a storage area on processor 110 for storing information, including data. It is understood that the aspects of the invention are the described intra-add/subtract instruction set and a code sequence for performing transformations for operating on packed data. According to these aspects of the invention, the storage area used for storing the packed data is not critical. Execution unit 120 is coupled to cache 140 and decoder 150. Cache 140 is used to cache data and/or control signals (such as instructions) from, for example, main memory 104. Decoder 150 is used for decoding instructions received by processor 110 into control signals and/or microcode entry points. In response to these control signals and/or microcode entry points, execution unit 120 performs the appropriate operations. Decoder 150 may be implemented using any number of different mechanisms (e.g., a look-up table, a hardware implementation, a PLA, etc.).

[0069] Figure 5a additionally shows a data storage device 108, (e.g., a magnetic disk, optical disk, and/or other machine readable media) can be coupled to computer system 100. In addition, the data storage device 108 is shown including code 195 for execution by the

processor 110. The code 195 can be written to cause the processor 110 to perform transformations with the intra-add/subtract instruction(s) for any number of purposes (e.g., motion video compression/decompression, image filtering, audio signal compression, filtering or synthesis, modulation/demodulation, etc.). Computer system 100 can also be coupled via bus 102 to a display device 170, a user input device 172, a hard copy device 176, a sound recording and/or playback device 178, a video digitizing device 180, and/or a communications device 190 (e.g., a serial communications chip, a wireless interface, an ethernet chip or a modem, which provides communications with an external device or network).

[0070] **Figure 5b** illustrates an alternative embodiment of a data processing system 200 which implements the principles of the present invention. One embodiment of data processing system 200 is an Intel® Personal Internet Client Architecture (Intel® PCA) applications processors with Intel XScale™ technology (as described on the world-wide web at developer.intel.com). It will be readily appreciated by one of skill in the art that the embodiments described herein can be used with alternative processing systems without departure from the scope of the invention.

[0071] Computer system 200 comprises a processing core 210 capable of performing SIMD operations including multiplications and horizontal additions and/or subtractions. For one embodiment, processing core 210 represents a processing unit of any type of architecture, including but not limited to a CISC, a RISC or a VLIW type architecture. Processing core 210 may also be suitable for manufacture in one or more process technologies and by being represented on a machine readable media in sufficient detail, may be suitable to facilitate said manufacture.

[0072] Processing core 210 comprises an execution unit 220, a set of register file(s) 230, and a decoder 250. Processing core 210 also includes additional circuitry (not shown) which is not necessary to the understanding of the present invention.

[0073] Execution unit 220 is used for executing instructions received by processing core 210. In addition to recognizing typical processor instructions, execution unit 220 recognizes instructions in packed instruction set 222 for performing operations on packed data formats. Packed instruction set 222 includes instructions for supporting intra-add/subtract operations, multiply operations, and may also include other packed instructions.

[0074] Execution unit 220 is coupled to register file 230 by an internal bus. Register file 230 represents a storage area on processing core 210 for storing information, including data. As previously mentioned, it is understood that the storage area used for storing the packed data is not critical. Execution unit 220 is coupled to decoder 250. Decoder 250 is used for decoding instructions received by processing core 210 into control signals and/or microcode entry points. In response to these control signals and/or microcode entry points, execution unit 220 performs the appropriate operations.

[0075] Processing core 210 is coupled with bus 214 for communicating with various other system devices, which may include but are not limited to, for example, synchronous dynamic random access memory (SDRAM) control 271, static random access memory (SRAM) control 272, burst flash memory interface 273, personal computer memory card international association (PCMCIA)/compact flash (CF) card control 274, liquid crystal display (LCD) control 275, direct memory access (DMA) controller 276, and alternative bus master interface 277.

[0076] In one embodiment, data processing system 200 may also comprise an I/O bridge 290 for communicating with various I/O devices via an I/O bus 295. Such I/O devices may include but are not limited to, for example, universal asynchronous receiver/transmitter (UART) 291, universal serial bus (USB) 292, Bluetooth wireless UART 293 and I/O expansion interface 294.

[0077] One embodiment of data processing system 200 provides for mobile, network and/or wireless communications and a processing core 210 capable of performing SIMD operations including intra-addition and/or subtraction. Processing core 210 may be programmed with various audio, video, imaging and communications algorithms including discrete transformations such as a Walsh-Hadamard transform, a fast Fourier transform (FFT), a discrete cosine transform (DCT), and their respective inverse transforms; compression/decompression techniques such as color space transformation, video encode motion estimation or video decode motion compensation; and modulation/demodulation (MODEM) functions such as pulse coded modulation (PCM).

[0078] **Figure 5c** illustrates alternative embodiments of a data processing system capable of performing SIMD intra-addition/subtraction operations. In accordance with one alternative embodiment, data processing system 300 may include a main processor 324, a SIMD coprocessor 326, a cache memory 340 and an input/output system 390. The input/output system 390 may optionally be coupled to a wireless interface 393. SIMD coprocessor 326 is capable of performing SIMD operations including multiplications and horizontal additions and/or subtractions. Processing core 310 may be suitable for manufacture in one or more process technologies and by being represented on a machine

readable media in sufficient detail, may be suitable to facilitate the manufacture of all or part of data processing system 300 including processing core 310.

[0079] For one embodiment, SIMD coprocessor 326 comprises an execution unit 320 and a set of register file(s) 330. One embodiment of main processor 324 comprises a decoder 350 to recognize instructions of instruction set 322 including SIMD multiply instructions and horizontal or intra-add/subtract instructions for execution by execution unit 320. For alternative embodiments, SIMD coprocessor 326 also comprises at least part of decoder 350b to decode instructions of instruction set 322. Processing core 310 also includes additional circuitry (not shown) which is not necessary to the understanding of the present invention.

[0080] In operation, the main processor 324 executes a stream of data processing instructions that control data processing operations of a general type including interactions with the cache memory 340, and the input/output system 390. Embedded within the stream of data processing instructions are SIMD coprocessor instructions. The decoder 350 of main processor 324 recognizes these SIMD coprocessor instructions as being of a type that should be executed by an attached SIMD coprocessor 326. Accordingly, the main processor 324 issues these SIMD coprocessor instructions (or control signals representing SIMD coprocessor instructions) on the coprocessor bus 236 where from they are received by any attached SIMD coprocessors. In this case, the SIMD coprocessor 326 will accept and execute any received SIMD coprocessor instructions intended for it.

[0081] Data may be received via wireless interface 393 for processing by the SIMD coprocessor instructions. For one example, voice communication may be received in the form of a digital signal, which may be processed by the SIMD coprocessor instructions to

regenerate digital audio samples representative of the voice communications. For another example, compressed audio and/or video may be received in the form of a digital bit stream, which may be processed by the SIMD coprocessor instructions to regenerate digital audio samples and/or motion video frames.

[0082] For one embodiment of processing core 310, main processor 324 and a SIMD coprocessor 326 are integrated into a single processing core 310 comprising an execution unit 320, a set of register file(s) 330, and a decoder 350 to recognize instructions of instruction set 322 including SIMD multiply instructions and horizontal or intra-add/subtract instructions for execution by execution unit 320.

OPERATION ENCODING FORMATS

[0083] Turning next to **Figure 6a**, in some alternative embodiments, 64 bit single instruction multiple data (SIMD) arithmetic operations may be performed through a coprocessor data processing (CDP) instruction. Operation encoding (opcode) format 401 depicts one such CDP instruction having CDP opcode fields 411 and 418. The type of CDP instruction, for alternative embodiments of horizontal or intra-add/subtract operations, may be encoded by one or more of fields 412, 413, 416 and 417. Up to three operand locations per instruction may be identified, including up to two source operand identifiers 402 and 403 and one destination operand identifier 405. One embodiment of the coprocessor can operate on 8, 16, 32, and 64 bit values. For one embodiment, the intra-addition/subtraction is performed on fixed-point or integer data values. For alternative embodiments, intra-addition/subtraction may be performed on floating-point data values. In some embodiments, the horizontal or intra-add/subtract instructions may be executed conditionally, using

condition field 410. For some horizontal or intra-add/subtract instructions source data sizes may be encoded by field 412.

[0084] In some embodiments of the horizontal or intra-add/subtract instructions, Zero (Z), negative (N), carry (C), and overflow (V) detection can be done on SIMD fields. Also, signed saturation and/or unsigned saturation to the SIMD field width may be performed for some embodiments of intra-add/subtract operations. In some embodiments of the horizontal or intra-add/subtract instructions in which saturation is enabled, saturation detection may also be done on SIMD fields. For some instructions, the type of saturation may be encoded by field 413. For other instructions, the type of saturation may be fixed.

[0085] **Figure 6b** is a depiction of an alternative operation encoding (opcode) format 501, having thirty-two or more bits, and register/memory operand addressing modes corresponding with a type of opcode format described in the "IA-32 Intel Architecture Software Developer's Manual Volume 2: Instruction Set Reference," which is which is available from Intel Corporation, Santa Clara, CA on the world-wide-web (www) at intel.com/design/litcentr. The type of intra-add/subtract operation, may be encoded by one or more of fields 512 and 514. Up to two operand locations per instruction may be identified, including up to two source operand identifiers 502 and 503. For one embodiment of the intra-add/subtract instruction, destination operand identifier 505 is the same as source operand identifier 502. For an alternative embodiment, destination operand identifier 505 is the same as source operand identifier 503. Therefore, for embodiments of the intra-add/subtract operations, one of the source operands identified by source operand identifiers 502 and 503 is overwritten by the results of the intra-add/subtract operations. For one

embodiment of the intra-add/subtract instruction, operand identifiers 502 and 503 may be used to identify 64-bit source and destination operands.

[0086] **Figure 6c** is a depiction of another alternative operation encoding (opcode) format 601, having forty or more bits. Opcode format 601 corresponds with opcode format 501 and comprises an optional prefix byte 610. The type of intra-add/subtract operation, may be encoded by one or more of fields 610, 612 and 614. Up to two operand locations per instruction may be identified by source operand identifiers 602 and 603 and by prefix byte 610. For example, in one embodiment of the intra-add instruction, field 612 may be set to a hexadecimal value of 0F38 and field 614 may be set to a hexadecimal value of 01 to indicate that data associated with source operand identifiers 602 and 603 are to be treated as signed packed words and result data associated with destination operand identifier 605 are to be treated as signed packed words. For an alternative embodiment of the intra-add instruction, field 614 may be set to a hexadecimal value of 03 to indicate that result data associated with destination operand identifier 605 are to be saturated to signed word values. For another alternative embodiment of the intra-add instruction, field 614 may be set to a hexadecimal value of 02 to indicate that data associated with source operand identifiers 602 and 603 are to be treated as signed packed doublewords and result data associated with destination operand identifier 605 are to be treated as signed packed doublewords. For one embodiment of the intra-subtract instruction, field 614 may be set to a hexadecimal value of 05 to indicate that data associated with source operand identifiers 602 and 603 are to be treated as signed packed words and result data associated with destination operand identifier 605 are to be treated as signed packed words. For an alternative embodiment of the intra-subtract instruction, field 614 may be set to a hexadecimal value of 07 to indicate that result data

associated with destination operand identifier 605 are to be saturated to signed word values. For another alternative embodiment of the intra-subtract instruction, field 614 may be set to a hexadecimal value of 06 to indicate that data associated with source operand identifiers 602 and 603 are to be treated as signed packed doublewords and result data associated with destination operand identifier 605 are to be treated as signed packed doublewords. For one embodiment of the intra-add/subtract instruction, prefix byte 610 may be used to identify 128-bit source and destination operands. For example, in one embodiment of the intra-add/subtract instruction, field 610 may be set to a hexadecimal value of 66 to indicate that 128 bits of data are associated with source operand identifiers 602 and 603 and 128 bits of result data are associated with destination operand identifier 605. For one embodiment of the intra-add/subtract instruction, destination operand identifier 605 is the same as source operand identifier 602. For an alternative embodiment, destination operand identifier 605 is the same as source operand identifier 603. Therefore, for embodiments of the intra-add/subtract operations, one of the source operands identified by source operand identifiers 602 and 603 is overwritten by the results of the intra-add/subtract operations.

[0087] Opcode formats 501 and 601 allow register to register, memory to register, register by memory, register by register, register by immediate, register to memory addressing specified in part by MOD fields 516 and 616 and by optional scale-index-base and displacement bytes.

DATA AND STORAGE FORMATS

[0088] **Figure 7a** illustrates alternative in-register fixed-point data storage formats. Each packed data includes more than one independent data element. Three packed data formats are illustrated; packed byte 711, packed word 712 and packed doubleword 713 together with quadword 714. One embodiment of packed byte 711 is sixty-four bits long containing eight data elements. Each data element is one byte long. One alternative embodiment of a packed byte format (not shown) is one hundred twenty-eight bits long containing sixteen byte data elements. Generally, a data element is an individual piece of data that is stored in a single register (or memory location) with other data elements of the same length. In one embodiment of the present invention, the number of data elements stored in a register is sixty-four bits divided by the length in bits of a data element. In an alternative embodiment of the present invention, the number of data elements stored in a register is one hundred twenty-eight bits divided by the length in bits of a data element.

[0089] One embodiment of packed word 712 is sixty-four bits long and contains four half word data elements. Each word data element in this embodiment contains sixteen bits of information.

[0090] One embodiment of packed doubleword 713 is sixty-four bits long and contains two doubleword data elements. Each doubleword data element in this embodiment contains thirty-two bits of information.

[0091] **Figure 7b** illustrates alternative in-register data storage formats. Each packed data includes more than one independent data element. Three packed data formats are illustrated; packed half 721, packed single 722 and packed double 723. One embodiment of packed half 721, packed single 722 and packed double 723 contain fixed-point data

elements. For an alternative embodiment one or more of packed half 721, packed single 722 and packed double 723 may contain floating-point data elements. One alternative embodiment of packed half 721 is one hundred twenty-eight bits long containing eight 16-bit data elements.

[0092] One embodiment of packed single 722 is one hundred twenty-eight bits long and contains four 32-bit data elements.

[0093] One embodiment of packed double 723 is one hundred twenty-eight bits long and contains two 64-bit data elements.

[0094] It will be appreciated that such packed data formats may be further extended to other register lengths, for example, to 96-bits, 160-bits, 192-bits, 224-bits, 256-bits or more.

DESCRIPTION OF SATURATION

[0095] As mentioned previously, some opcode embodiments may indicate whether intra-addition/subtraction operations optionally saturate. Where the result of an operation, with saturate enabled, overflows or underflows the range of the data, the result will be clamped. Clamping means setting the result to a maximum or minimum value should a result exceed the range's maximum or minimum value. In the case of underflow, saturation clamps the result to the lowest value in the range and in the case of overflow, to the highest value. The allowable range for each data format of one embodiment is shown in Table 2.

TABLE 2

Data Format	Minimum Value	Maximum Value
Unsigned Byte	0	255
Signed Byte	-128	127
Unsigned word	0	65535
Signed word	-32768	32767
Unsigned Doubleword	0	$2^{32}-1$
Signed Doubleword	-2^{31}	$2^{31}-1$
Unsigned Quadword	0	$2^{64}-1$
Signed Quadword	-2^{63}	$2^{63}-1$

[0096] Therefore, using the unsigned byte data format, if an operation's result=258 and saturation was enabled, then the result would be clamped to 255 before being stored into the operation's destination register. Similarly, if an operation's result=-32999 and a signed word data format with saturation enabled, then the result would be clamped to -32768 before being stored into the operation's destination register.

[0097] With no saturation, only the lower bits of the result are presented. With unsigned saturation, the bits from zero to the maximum unsigned value may be presented. With signed saturation, bits from the maximum positive to the maximum negative values are presented.

INTRA-ADD/SUBTRACT OPERATION(S)

[0098] In one embodiment of the invention, the SRC1 register contains packed data (Source1), the SRC2 register contains packed data (Source2) and the DEST register will contain the result (Result) of performing the horizontal add instruction on Source1 and Source2. In the first step of the horizontal add/subtract instruction, one or more pairs of data elements from Source1 are summed/subtracted together. Similarly, one or more pairs of data

elements from Source2 are summed/subtracted together. The results of the instruction are then stored in the DEST register.

[0099] **Figure 8a** is a flow diagram illustrating one embodiment of a process 800 for performing the intra-add operation of Figure 3a. Process 800 and other processes herein disclosed are performed by processing blocks that may comprise dedicated hardware or software or firmware operation codes executable by general purpose machines or by special purpose machines or by a combination of both.

[0100] Process 800 begins from a start state and proceeds to processing block 802 where a control signal is decoded. In particular, the control signal identifies an operation code of a horizontal add or intra-add instruction. The process 800 then advances to processing block 804, where the registers in a register file or a memory are accessed at locations specified by the SRC1 and SRC2 addresses. The register file or memory provides an execution unit with the packed data (Source1) stored in at the SRC1 address, and the packed data (Source2) stored at the SRC2 address.

[0101] Process 800 proceeds to processing block 806, where the execution unit is enabled to perform the intra-add operation. Next, process 800 advances to processing block 808, where the intra-add operation of Figure 3a is performed. Source1 bits thirty-one through zero are added to Source1 bits sixty-three through thirty-two, generating a first 32-bit result (Result[31:0]). Source1 bits ninety-five through sixty-four are added to Source1 bits one hundred-and-twenty-seven through ninety-six, generating a second 32-bit result (Result[63:32]). Source2 bits thirty-one through zero are added to Source2 bits sixty-three through thirty-two, generating a third 32-bit result (Result[95:64]). Source2 bits ninety-five

through sixty-four are added to Source1 bits one hundred-and-twenty-seven through ninety-six, generating a fourth 32-bit result (Result[127:96]).

[0102] The process 800 advances to processing block 810, where the results of the intra-add instruction are stored in a register in a register file or a memory at the DEST address. The process 800 then terminates. Of course, the process of Figure 8a can be easily altered to describe the horizontal addition of other packed data formats.

[0103] **Figure 8b** is a flow diagram illustrating one embodiment of a process 820 for performing the intra-subtract operation of Figure 3b. Process 820 begins from a start state and proceeds to processing block 802 where, again, a control signal is decoded. In particular, the control signal identifies an operation code of a horizontal subtract or intra-subtract instruction. The process 820 then advances to processing block 804, where the registers in a register file or a memory are accessed at locations specified by the SRC1 and SRC2 addresses. The register file or memory provides an execution unit with the packed data (Source1) stored in at the SRC1 address, and the packed data (Source2) stored at the SRC2 address.

[0104] Process 800 proceeds to processing block 806, where the execution unit, in accordance with the instruction decoded, is enabled to perform the intra-subtract operation. Next, process 800 advances to processing block 828, where the intra-subtract operation of Figure 3b is performed. Source1 bits sixty-three through thirty-two are subtracted from Source1 bits thirty-one through zero, generating a first 32-bit result (Result[31:0]). Source1 bits one hundred-and-twenty-seven through ninety-six are subtracted from Source1 bits ninety-five through sixty-four, generating a second 32-bit result (Result[63:32]). Source2 bits sixty-three through thirty-two are subtracted from Source2 bits thirty-one through zero,

generating a third 32-bit result (Result[95:64]). Source2 bits one hundred-and-twenty-seven through ninety-six are subtracted from Source2 bits ninety-five through sixty-four, generating a fourth 32-bit result (Result[127:96]).

[0105] The process 820 advances to processing block 810, where the results of the intra-subtract instruction are stored in a register in a register file or a memory at the DEST address. The process 820 then terminates.

[0106] **Figure 8c** is a flow diagram illustrating an alternative embodiment of a process 830 for performing the intra-add operation. Process 830 begins from a start state and proceeds to processing block 802 where a control signal is decoded. In particular, the control signal identifies an operation code of an intra-add instruction. In processing block 804 the registers in a register file or a memory are accessed according to the SRC1 and SRC2 addresses. The register file or memory provides the execution unit with Source1 stored in at the SRC1 address, and Source2 stored at the SRC2 address.

[0107] In processing block 806, the execution unit is enabled to perform the intra-add operation. Next, in processing block 838, Source1 bits thirty-one through sixteen are added to Source1 bits fifteen through zero, generating a first 16-bit result (Result[15:0]). Source1 bits sixty-three through forty-eight are added to Source1 bits one forty-seven through thirty-two, generating a second 16-bit result (Result[31:16]). Source1 bits ninety-five through eighty are added to Source1 bits seventy-nine through sixty-four, generating a third 16-bit result (Result[47:32]). Source1 bits one hundred-and-twenty-seven through one hundred-and-twelve are added to Source1 bits one hundred-and-eleven through ninety-six, generating a fourth 16-bit result (Result[63:48]). Source2 bits thirty-one through sixteen are added to Source2 bits fifteen through zero, generating a fifth 16-bit result (Result[79:64]). Source2

bits sixty-three through forty-eight are added to Source2 bits one forty-seven through thirty-two, generating a sixth 16-bit result (Result[95:80]). Source2 bits ninety-five through eighty are added to Source2 bits seventy-nine through sixty-four, generating a seventh 16-bit result (Result[111:96]). Source2 bits one hundred-and-twenty-seven through one hundred-and-twelve are added to Source2 bits one hundred-and-eleven through ninety-six, generating an eighth 16-bit result (Result[127:112]).

[0108] The process 830 advances to processing block 810, where the results of the intra-add instruction are stored in a register in a register file or a memory at the DEST address. The process 830 then terminates.

[0109] **Figure 8d** is a flow diagram illustrating an alternative embodiment of a process 840 for performing the intra-subtract operation. Process 840 begins from a start state and proceeds to processing block 802 where a control signal is decoded. In particular, the control signal identifies an operation code of an intra-subtract instruction. In processing block 804 the registers in a register file or a memory are accessed according to the SRC1 and SRC2 addresses. The register file or memory provides the execution unit with Source1 stored in at the SRC1 address, and Source2 stored at the SRC2 address.

[0110] In processing block 806, the execution unit is enabled to perform the intra-subtract operation. Next, in processing block 848, Source1 bits thirty-one through sixteen are subtracted from Source1 bits fifteen through zero, generating a first 16-bit result (Result[15:0]). Source1 bits sixty-three through forty-eight are subtracted from Source1 bits one forty-seven through thirty-two, generating a second 16-bit result (Result[31:16]). Source1 bits ninety-five through eighty are subtracted from Source1 bits seventy-nine through sixty-four, generating a third 16-bit result (Result[47:32]). Source1 bits one

hundred-and-twenty-seven through one hundred-and-twelve are subtracted from Source1 bits one hundred-and-eleven through ninety-six, generating a fourth 16-bit result (Result[63:48]). Source2 bits thirty-one through sixteen are subtracted from Source2 bits fifteen through zero, generating a fifth 16-bit result (Result[79:64]). Source2 bits sixty-three through forty-eight are subtracted from Source2 bits one forty-seven through thirty-two, generating a sixth 16-bit result (Result[95:80]). Source2 bits ninety-five through eighty are subtracted from Source2 bits seventy-nine through sixty-four, generating a seventh 16-bit result (Result[111:96]). Source2 bits one hundred-and-twenty-seven through one hundred-and-twelve are subtracted from Source2 bits one hundred-and-eleven through ninety-six, generating an eighth 16-bit result (Result[127:112]).

[0111] The process 840 advances to processing block 810, where the results of the intra-add instruction are stored in a register in a register file or a memory at the DEST address. The process 840 then terminates.

[0112] **Figure 8e** is a flow diagram illustrating one embodiment of a process 850 for performing the intra-add operation of Figure 3c. Process 850 begins from a start state and proceeds to processing block 802 where a control signal that identifies an operation code of an intra-add instruction is decoded. The process 850 then advances to processing block 804, where the registers in a register file or a memory are accessed and the execution unit is supplied with the packed data (Source1) stored in at the SRC1 address, and the packed data (Source2) stored at the SRC2 address.

[0113] Process 850 proceeds to processing block 806, where the execution unit is enabled to perform the intra-add operation. Next, in processing block 858, the intra-add operation of Figure 3c is performed. Source1 bits thirty-one through zero are added to

Source1 bits sixty-three through thirty-two, generating a first 32-bit result (Result[31:0]). Source2 bits thirty-one through zero are added to Source2 bits sixty-three through thirty-two, generating a second 32-bit result (Result[63:32]).

[0114] The process 850 advances to processing block 810, where the results of the intra-add instruction are stored in a register in a register file or a memory at the DEST address. The process 850 then terminates.

[0115] **Figure 8f** is a flow diagram illustrating another alternative embodiment of a process 860 for performing the intra-subtract operation. Process 860 begins from a start state and proceeds to processing block 802 where, a control signal that identifies an operation code of an intra-subtract instruction is decoded. The process 860 then advances to processing block 804, where the registers in a register file or a memory are accessed and the execution unit is supplied with the packed data (Source1) stored in at the SRC1 address, and the packed data (Source2) stored at the SRC2 address.

[0116] Process 860 proceeds to processing block 806, where the execution unit, in accordance with the instruction decoded, is enabled to perform the intra-subtract operation. Next, in processing block 868, Source1 bits sixty-three through thirty-two are subtracted from Source1 bits thirty-one through zero, generating a first 32-bit result (Result[31:0]). Source2 bits sixty-three through thirty-two are subtracted from Source2 bits thirty-one through zero, generating a second 32-bit result (Result[63:32]).

[0117] The process 860 advances to processing block 810, where the results of the intra-subtract instruction are stored in a register in a register file or a memory at the DEST address. The process 860 then terminates.

[0118] **Figure 8g** is a flow diagram illustrating an alternative embodiment of a process 870 for performing the intra-add operation of Figure 3a. Process 870 begins from a start state and proceeds to processing block 802 where a control signal that identifies an operation code of an intra-add instruction is decoded. The process 870 then advances to processing block 804, where the registers in a register file or a memory are accessed and the execution unit is supplied with the packed data (Source1) stored in at the SRC1 address, and the packed data (Source2) stored at the SRC2 address.

[0119] In processing block 806, the execution unit is enabled to perform the intra-add operation of Figure 3a. Next, in processing block 878, Source1 bits thirty-one through sixteen are added to Source1 bits fifteen through zero, generating a first 16-bit result (Result[15:0]). Source1 bits sixty-three through forty-eight are added to Source1 bits one forty-seven through thirty-two, generating a second 16-bit result (Result[31:16]). Source2 bits thirty-one through sixteen are added to Source2 bits fifteen through zero, generating a third 16-bit result (Result[47:32]). Source2 bits sixty-three through forty-eight are added to Source2 bits one forty-seven through thirty-two, generating a fourth 16-bit result (Result[63:48]).

[0120] The process 870 advances to processing block 810, where the results of the intra-add instruction are stored in a register in a register file or a memory at the DEST address. The process 870 then terminates.

[0121] **Figure 8h** is a flow diagram illustrating an alternative embodiment of a process 880 for performing the intra-subtract operation of Figure 3b. Process 880 begins from a start state and proceeds to processing block 802 where a control signal that identifies an operation code of an intra-subtract instruction is decoded. The process 880 then advances to

processing block 804, where the registers in a register file or a memory are accessed and the execution unit is supplied with the packed data (Source1) stored in at the SRC1 address, and the packed data (Source2) stored at the SRC2 address.

[0122] In processing block 806, the execution unit is enabled to perform the intra-add operation of Figure 3b. Next, in processing block 888, Source1 bits thirty-one through sixteen are subtracted from Source1 bits fifteen through zero, generating a first 16-bit result (Result[15:0]). Source1 bits sixty-three through forty-eight are subtracted from Source1 bits one forty-seven through thirty-two, generating a second 16-bit result (Result[31:16]). Source2 bits thirty-one through sixteen are subtracted from Source2 bits fifteen through zero, generating a third 16-bit result (Result[47:32]). Source2 bits sixty-three through forty-eight are subtracted from Source2 bits one forty-seven through thirty-two, generating a fourth 16-bit result (Result[63:48]).

[0123] The process 870 advances to processing block 810, where the results of the intra-add instruction are stored in a register in a register file or a memory at the DEST address. The process 870 then terminates. It will be appreciated that any of the processes of Figs. 8a-8h may be performed with signed saturation, with unsigned saturation or without saturation.

EXEMPLARY INTRA-ADD/SUBTRACT CIRCUIT(S)

[0124] In one embodiment, the intra-add/subtract instructions can execute on multiple data elements in the same number of clock cycles as an inter-add operation on unpacked data. To achieve execution in the same number of clock cycles, parallelism is used.

[0125] **Figure 9a** illustrates one embodiment of a circuit for performing horizontal or intra-add/subtract operations. Operation control 910 processes the control signal for the

intra-add operations. Operation control 910 outputs signals via signal line(s) 920 to control intra-adder 930.

[0126] The intra-adder 930 receives inputs from Source1[127:0], Source2[127:0], and Enable 920. The intra-adder 930 includes four adder circuits 932, 934, 936 and 938. Adder 932 receives inputs from Source2[127:64], adder 934 receives inputs from Source2[63:0], adder 936 receives inputs from Source1[127:64], while adder 938 receives inputs from Source1[63:0]. When enabled, the adders 932, 934, 936 and 938 sum their respective inputs, and each generates a 32-bit output. The results of the addition by adder 932 (i.e., Result[127:96]), adder 934 (i.e., Result[95:64], by adder 936 (i.e., Result[63:32]), and by adder 938 (i.e., Result[31:0]) are combined into the 128-bit Result and communicated to the Result Register 940.

[0127] **Figure 9b** illustrates an alternative embodiment of a circuit for performing horizontal or intra-add/subtract operations. Operation control 910 processes the control signal for the intra-add/subtract operations. Operation control 910 outputs signals via signal line(s) 920 to control adders 931 and 933, multiplexers m0-mF, multiplexers 939 and saturation circuitry 940. Through the use of multiplexers m0-mF, operand data elements from Source1 and Source2 may be aligned to positions at the inputs of adders 931 and 933 to facilitate horizontal or intra-addition/subtraction as well as to facilitate vertical or inter-addition/subtraction. Through the use of multiplexers 939, the results form adders 931 and 933 may be aligned to facilitate intra-addition/subtraction of different width operands, Source1 and Source2. It will be appreciated that various sizes of operand data elements and of operands Source1 and Source2 may be supported through such an approach.

[0128] For example, one embodiment of intra-add/subtract operations as illustrated in Figures 3a and 3b and as described in Figures 8g and 8h, respectively, uses 16-bit data elements 914-911 of Source1 and 924-921 of Source2. To accomplish such an operation through the use of adder 933, operation control 910 signals for the outputs of multiplexers m3-m0 to be elements 924, 922, 914 and 912 respectively. Operation control 910 signals for the outputs of multiplexers mB-m8 to be elements 923, 921, 913 and 911 respectively. In this way adder 933 can perform a 16-bit intra-addition/subtraction in substantially the same manner as a 16-bit inter-addition/subtraction, producing element 923 plus/minus element 924, element 921 plus/minus element 922, element 913 plus/minus element 914, and element 911 plus/minus element 912. Operation control 910 also signals for the multiplexers 939 to pass through results from the adders without realigning them and so the results may be optionally saturated and stored as elements 944-941 respectively.

[0129] An alternative embodiment of intra-add/subtract operations as illustrated in Figure 3c and as described in Figures 8e and 8f, respectively, uses 32-bit data elements 914-913, and 912-911 of Source1 and 924-923, and 922-921 of Source2. To accomplish such an operation through the use of adder 933, operation control 910 signals for the outputs of multiplexers m3-m0 to be elements 924, 923, 914 and 913 respectively. Operation control 910 signals for the outputs of multiplexers mB-m8 to be elements 922, 921, 912 and 911 respectively. In this way adder 933 can perform a 32-bit intra-addition/subtraction in substantially the same manner as a 32-bit inter-addition/subtraction, producing element 922-921 plus/minus element 924-923, and element 912-911 plus/minus element 914-913. Once again operation control 910 signals for the multiplexers 939 to pass through results from the

adders without realigning them and so the results may be optionally saturated and stored as two 32-bit elements 944-943 and 942-941 respectively.

[0130] Another alternative embodiment of intra-add/subtract operations as illustrated in Figures 3a and 3b and as described in Figures 8a and 8b, respectively, uses four 32-bit elements 918-917, 916-915, 914-913, and 912-911 of Source1 and 928-927, 926-925, 924-923, and 922-921 of Source2. To accomplish such an operation through the use of adders 931 and 933, operation control 910 signals for the outputs of multiplexers m7-m0 to be elements 928, 927, 918, 917, 924, 923, 914 and 913 respectively. Operation control 910 signals for the outputs of multiplexers mF-m8 to be elements 926, 925, 916, 915, 922, 921, 912 and 911 respectively. In this way adders 931 and 933 can perform 32-bit intra-additions/subtractions in substantially the same manner as 32-bit inter-additions/subtractions, producing element 926-925 plus/minus element 928-927, element 916-915 plus/minus element 918-917, element 922-921 plus/minus element 924-923, and element 912-911 plus/minus element 914-913. For the 128-bit operands, operation control 910 signals for the multiplexers 939 to realign the two middle results from the adders by swapping a result from adder 931 (bits 95-64) with a result from adder 933 (bits 63-32) to produce element 926-925 plus/minus element 928-927, element 922-921 plus/minus element 924-923, element 916-915 plus/minus element 918-917, and element 912-911 plus/minus element 914-913. These results may be optionally saturated and stored as four 32-bit elements 948-947 and 946-945 944-943 and 942-941 respectively.

[0131] It will be appreciated that multiplexers 939 may provide timing advantages by reducing some of the wiring delay associated with 128-bit operands and reducing multiplexer complexity before the adder. It will also be appreciated that techniques such as

those illustrated by Figure 9b may be modified in arrangement or detail without departing from the broader spirit of the invention.

[0132] Another embodiment of intra-add/subtract operations as described in Figures 8c and 8d, respectively, uses 16-bit data elements 918-911 of Source1 and 928-921 of Source2. To accomplish such an operation through the use of adders 931 and 933, operation control 910 signals for the outputs of multiplexers m7-m0 to be elements 928, 926, 918, 916, 924, 922, 914 and 912 respectively. Operation control 910 signals for the outputs of multiplexers mF-m8 to be elements 927, 925, 917, 915, 923, 921, 913 and 911 respectively. In this way adders 931 and 933 can perform 16-bit intra-additions/subtractions in substantially the same manner as 16-bit inter-additions/subtractions, producing element 927 plus/minus element 928, element 925 plus/minus element 926, element 917 plus/minus element 918, element 915 plus/minus element 916, 923 plus/minus element 924, element 921 plus/minus element 922, element 913 plus/minus element 914, and element 911 plus/minus element 912. For the 128-bit operands, operation control 910 signals for the multiplexers 939 to realign the four middle results from the adders by swapping two results from adder 931 (bits 95-64) with two results from adder 933 (bits 63-32) to produce element 927 plus/minus element 928, element 925 plus/minus element 926, 923 plus/minus element 924, element 921 plus/minus element 922, element 917 plus/minus element 918, element 915 plus/minus element 916, element 913 plus/minus element 914, and element 911 plus/minus element 912. These results may be optionally saturated and stored as 16-bit elements 948-941 respectively.

[0133] For performing vertical or inter-addition/subtraction operations, operation control 910 signals for the outputs of multiplexers m7-m0 to be elements 918-911 respectively and

for the outputs of multiplexers mF-m8 to be elements 928-921 respectively. Operation control 910 signals for the multiplexers 939 to pass through results from the adders without realigning them and so the results may be optionally saturated according to the operation and stored as 948-941 respectively.

CONCLUSION

[0134] The horizontal intra-add operation facilitates the efficient performance of butterfly computations. It further increases code density by eliminating the need for the rearrangement of data elements and the corresponding rearrangement operations. With the addition of some novel control and multiplexer circuitry, intra-add/subtract operations may be performed with the same arithmetic circuitry as inter-add/subtract operations.

[0135] The present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. The described embodiments are to be considered in all respects only as illustrative and not restrictive. The scope of the invention is, therefore, indicated by the appended claims rather than the foregoing description. All changes which come within the meaning and range of equivalency of the claims are to be embraced within their scope.